TDA Data Management Planning: Readability of Flow Charts

E. C. Posner
TDA Planning Office

This article proposes a "global" readability standard for flow charts in DSN software implementations. The standard limits the kinds of closures and returns that can occur from decision nodes. It is proved that the standard is equivalent to permitting only those flow charts that are constructible from hierarchical expansion of the three structures BLOCK, IFTHENELSE, and LOOPREPEAT. The LOOPREPEAT structure is the simultaneous generalization of DOWHILE and DOUNTIL. Considerations of code as opposed to flow chart readability, however, may rule out the use of LOOPREPEAT in favor of allowing only its special cases DOWHILE and DOUNTIL.

I. Introduction

We propose a standard for flow charts to be used in DSN implementations. We first define a flow chart as a finite directed graph with five kinds of nodes:

(1) START nodes with no links entering the node and one link leaving.

(2) END nodes with one link entering and none leaving.

(3) FUNCTION nodes with one link entering and one leaving.

(4) DECISION nodes with one link entering and two leaving.

(5) COLLECTOR nodes with two links entering and one leaving.

We further require that there be exactly one START node and one END node. We also demand that the flow chart be *connected* in the sense that, given any node, there is a directed path from START to END going through the node. The condition on DECISION nodes that they have only two outputs is a mathematical convenience for proof purposes, and the use of multiple output decisions (cases) would be allowed in any DSN standard. A similar comment applies to COLLECTOR nodes.

More precisely, a (finite) directed graph is a collection of points called *nodes*, and directed edges, called *links*. Nodes are connected to nodes by links, where the direction is indicated by an arrowhead at the *forward* end of the link, the other end being called the *rear*. Each link must start and end at a (possibly the same) node. It starts

at its rear end and ends at its forward end. A *path* from one node n_1 to another n_2 is a sequence of nodes $n_1 = m_1$, m_2, $--$, $m_p = n_2$ and links ℓ_1, ℓ_2, $--$, ℓ_{p-1}, such that link ℓ_j starts at m_j and ends at m_{j+1}, $1 \leq j \leq p-1$.

A flow chart then is a directed graph, where the graph nodes are the function boxes, decision boxes, and collector nodes. We will, however, draw all nodes as dots, since the definition of "flow chart graph" makes it clear whether a given graph node corresponds to a function, decision, or collection.

Given flow charts G_1 and G_2, together with a function node F of G_1, we define the *hierarchical expansion of* G_1 *by replacing* F *by* G_2, $H(G_1, F; G_2)$ as a new flow chart G_3. Specifically, let link ℓ_1 end at F and ℓ_2 start at F. Remove node F from the set of nodes of G_1 and add the set of nodes of G_2 other than START and END. Let ℓ_1 end at the first node of G_2 reached from its START, and let ℓ_2 start at the last node before the END of G_2. Then the START and END of G_2 have been removed, but all other nodes remain and are the same kind of node. The fact that G_3 is a flow chart can be readily verified.

Let G be a flow chart and F a function node of G. We can get a new flow chart G' with one less node and one less link as follows: Remove F from the set of nodes of G. Let ℓ_1 end at F_1 and ℓ_2 start there. Remove these two links from the set of links and add a new link ℓ' which starts at the node from which ℓ_1 started, and ends where ℓ_2 ended (see Fig. 1). This operation permits us to remove a "no-op" function at a FUNCTION node.

Let A be a finite set of flow charts, called *structures*. Define $A_0 = A$, and A_1, the *one-step completion of* A, as the class of all flow charts which can be obtained from the structures in A by removing a function node, or by replacing a function node of a structure G by a structure H in A, where H may be different from G. Define A_n for $n > 1$, the *n-step completion of* A, to be the one-step completion of A_{n-1}. Define A_∞, *the completion of* A, to be

$$\bigcup_{n=1}^{\infty} A_n,$$

the flow charts obtainable from the structures in A by hierarchical expansion. Note that if G and H are in A_∞, and F is a function node of G, then replacing F in G by H leads to a flow chart still in A_∞; that is, A_∞ is *complete* under hierarchical expansion.

Finally, let us define a looping decision and a nonlooping decision. A *nonlooping decision* node in a flow chart is one for which all paths to END starting at the decision ultimately coincide at some collector node before any path from the decision again reaches that decision node. Otherwise the decision node is called a *looping decision.* A link starting at a decision node with the property that all paths to END starting with the link avoid the given decision node is a *non-looping link;* if there is a way to reach the decision node starting with the link, it is called a *looping link.*

Let us now define a set R of structures which consists of the three structures BLOCK, IFTHENELSE, and LOOPREPEAT, as in Fig. 2. Note that LOOPREPEAT becomes DOWHILE if node G is removed and DOUNTIL if F is removed. Hence, by Mills' Theorem (Ref. 1), every algorithm can be flow charted by a flow chart in R_∞, since it can be flow charted by a flow chart in the completion of BLOCK, IFTHENELSE, and DOWHILE (or alternatively of BLOCK IFTHENELSE, and DOUNTIL).

II. Readability of Flow Charts

A nonlooping decision node is said to have the *Unique Merger Property* if the node at which all paths out of the decision first merge has the property that any two paths starting with different links out of the decision node also merge for the first time at that node. The decision node in IFTHENELSE satisfies Unique Merger.

A looping decision node is said to have the *Forced Loop Closure Property* provided the following holds. An *external input* to a loop is a collector node reachable from the decision that has an output path to the decision node. It also has an input path from START not containing the same output link of the given decision node that was on the input path to the collector node. Note that each looping decision node has at least one external input to a loop it creates. We require that the following hold for at least (it turns out exactly) one of the output links of the looping decision; such links are called *looping links:* every path starting at a looping link must return to the looping decision. We require that there be only one external input node. We also demand that every path starting with the looping link go through the external input, and go through it on only one of its two input links, before returning to the looping decision. It follows from connectivity that only one of the two outputs of a looping decision can be looping—otherwise there is no way to reach END starting from the decision. Note that the decision node of LOOPREPEAT has the Forced Loop Closure Property.

We say that a flow chart has the *Readability Property* provided every nonlooping decision satisfies Unique Merger, and every looping decision satisfies Forced Loop Closure. Observe that if G_1 and G_2 satisfy the Readability Property, or, more briefly, are *Readable*, then if F is any function node in G_1, the flow chart $H(G_1, F; G_2)$ is also Readable. Thus every flow chart in R_∞ is Readable. The next section proves the main result of this article, that every Readable flow chart is in R_∞. Hence, the Readable flow charts can be obtained by using the three structures in R together with hierarchical expansion, and every flow chart so obtained is Readable. By Mills' Theorem previously referenced, then, we also conclude that every algorithm can be flow charted by a Readable flow chart.

III. The Main Theorem on Readability

The theorem below is proved by contradiction, but the proof is actually a recursive procedure for hierarchically expanding a Readable flow chart using the three structures in R: BLOCK, IFTHENELSE, and LOOPREPEAT.

THEOREM. *R_∞ is exactly the class of Readable flow charts.*

PROOF. That every flow chart in R_∞ is Readable has already been noted in the previous section. The hard part is to prove that every Readable flow chart can be obtained from hierarchical expansion of the three structures in R.

If not, let G be an alleged counterexample with the smallest number of nodes among the counterexamples, that is, among the Readable flow charts not in R_∞. Note that G has no function nodes, for they could be removed to yield another counterexample. We will show that G has no looping decisions. Let p be such a looping decision, with external input C as in Fig. 3.

Let π_1 denote the set of paths from C to p, and π_2 the set from p to C. Can there be a path λ from a node r on a path in π_2 to a node D on a path in π_1? No, because all paths from p starting with the looping link of p go through C, by the definition of Forced Loop Closure for the loop started by p.

Can there be a path μ from a node s on a path in π_1 to a node E on a path in π_2? The answer is again No, but for a slightly more complicated reason. This time, look at the loop started by s, which must satisfy Forced Loop Closure. The node C is still an external input node, but so is E because of the path $C \rightarrow s \rightarrow p \rightarrow E$. This situation is of course ruled out, so μ does not exist either.

Then Fig. 3 can be thought of as Fig. 4, where there are no paths between the nodes in A and in B (which each might be empty sets), and where there are no entries or exits from A and B other than the ones shown. That is, the original G is a hierarchical expansion (including possibly node removal) of a graph G_1 which has a function node in place of the structure of Fig. 4. Figure 4 itself is a hierarchical expansion of IFTHENELSE, the two graphs corresponding to A and B being the graphs replacing the FUNCTION nodes of LOOPREPEAT. These two graphs are also Readable, however, and have fewer nodes; hence they are in R_∞ (or are null). The graph G_1 as we have observed can be obtained from a Graph G_2 by hierarchically expanding at a function node by IFTHENELSE. But G_2 has fewer nodes than G_1, hence fewer nodes than G, and so is in R_∞. So G itself would be in R_∞. This proves that G has no looping decision nodes.

So G has only nonlooping decisions. Let p be such a node, with merger node M. An *external input* is a node C on a path from p to M that can be reached from START without going through p, or, if it cannot be reached from START without going through p, then such paths must go through M before reaching C. If we knew that there were no external inputs, we would be done as in the LOOPREPEAT case, for Unique Merger plus the lack of external inputs would cause the nonlooping decision to look like Fig. 5. The same hierarchical expansion idea would work.

First note that if C were an external input, there is no path from M to C not going through p. For it must be possible to reach END from M, and it must therefore be possible to reach END from M without going through C. Let r be a node at which a path to END first leaves the path from M to C. This r is, of course, a decision node, but it is looping because of the path $r \rightarrow C \rightarrow M \rightarrow r$. Since G has no looping decision nodes, there is no path from M to C not going through p.

Thus, if C exists at all, the situation of Fig. 6 must prevail: there is a path from START to C not going through p.

There is of course a path to p from START also. There are two possibilities: *i*) there is a path from START to p not going through C; *ii*) every path from START to p goes through C.

In case *i*), there is a last node s where the paths from START to C and to p agree, as in Fig. 7. However, s

violates Unique Merger, since paths starting with both links out of s meet for the first time at C and also at M. So we are in case *ii*), in which every path from START to p goes through C, as in Fig. 8. In this case, there is a path from M to p, for there is no way of leaving the path from C to M without going through M, by Unique Merger. Then p is a looping decision, already ruled out. This proves the theorem.

IV. Reversing a Flow Chart

This section proves an amusing corollary to the theorem of Section III. Some people read flow charts backward in trying to understand them, so define the *reversal* of a flow chart as the flow chart with the same nodes but arrowheads reversed. Then FUNCTION nodes remain FUNCTION nodes, DECISION nodes become COLLECTOR nodes, COLLECTOR nodes become DECISION nodes, and START and END are interchanged. However, we still have a flow chart, as is easy to see. If G is a flow chart, let G^R denote its reverse. Note the commutativity of reversing and hierarchical expansion:

$$H(G_1^R, F; G_2^R) = H^R(G_1, F; G_2)$$

Likewise, note the commutativity of reversing and completion:

$$(A^R)_\infty = (A_\infty)^R$$

We then have the following Corollary to the theorem of Section III.

COROLLARY. The reverse of a readable flow chart is a readable flow chart.

PROOF. By the theorem, if G is readable, G is in R_∞, where R consists of BLOCK, IFTHENELSE, and LOOPREPEAT. Hence G^R is in $(R^R)_\infty$. But the reverse of BLOCK is BLOCK, of IFTHENELSE is IFTHENELSE, and of LOOPREPEAT is LOOPREPEAT. So $R^R = R$, and G^R is in R_∞. By the theorem again, G^R is readable, as required.

V. Readability of Code

We have seen that if one adopts the flow chart readability requirement, the only permitted structures automatically become BLOCK, IFTHENELSE, and LOOPREPEAT. Therefore, those are all we would ever propose to even consider permitting as a DSN standard set of structures. It may however be preferable to not allow the full force of LOOPREPEAT but only permit DOWHILE and DOUNTIL, for reasons of code readability. The reason would be that the code for LOOPREPEAT is less readable.

The reason is that the EXIT from the loop of Fig. 9 occurs in the middle of the code, looking like Fig. 10. On the other hand, DOWHILE and DOUNTIL have their EXITS at the end of their code, as is proper for top-down readability (Fig. 11). But the exact form of the code and the tradeoffs involved seem rather language dependent, and the problem of whether to universally ban LOOPREPEAT for DSN software implementations is still under investigation.

Acknowledgment

The author is indebted to R. C. Tausworthe of the DSN Data Systems Development Section and M. D. Donner for valuable comments.

Reference

1. Mills, H. D., *Mathematical Foundations for Structural Programming*, IBM Federal Systems Div., Rockville, Md., February 1972.

Fig. 1. Removing a FUNCTION node

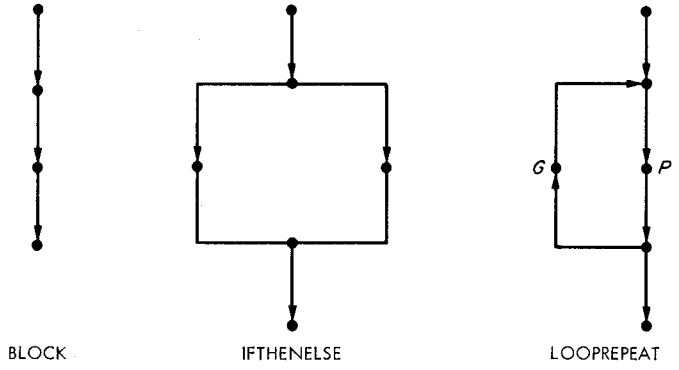

Fig. 2. The three structures of R

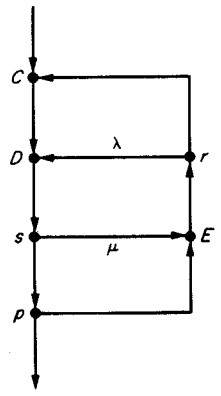

Fig. 3. A looping decision node in G

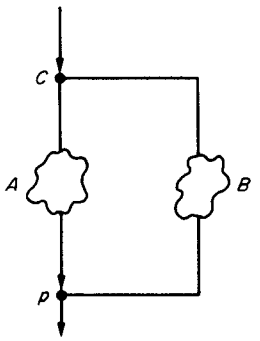

Fig. 4. How to view Fig. 3

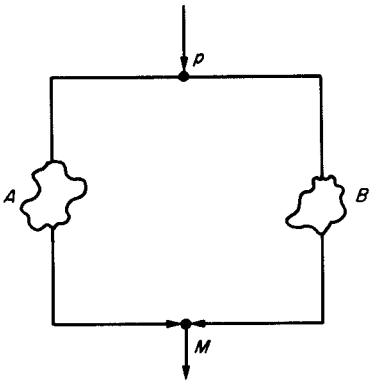

Fig. 5. A nonlooping decision

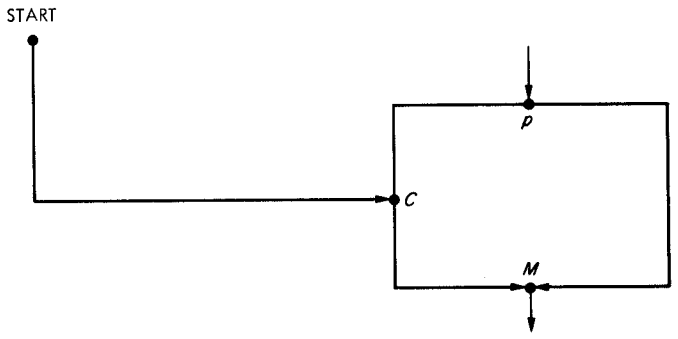

Fig. 6. The external input

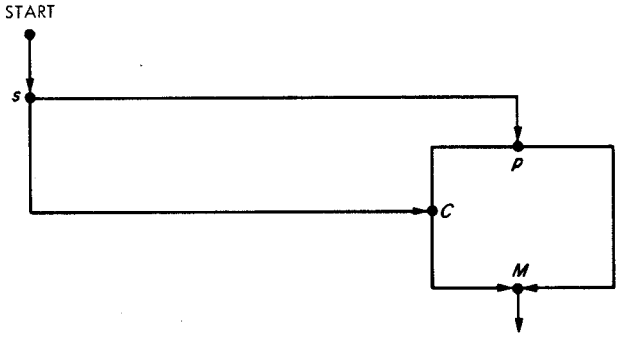

Fig. 7. The first case

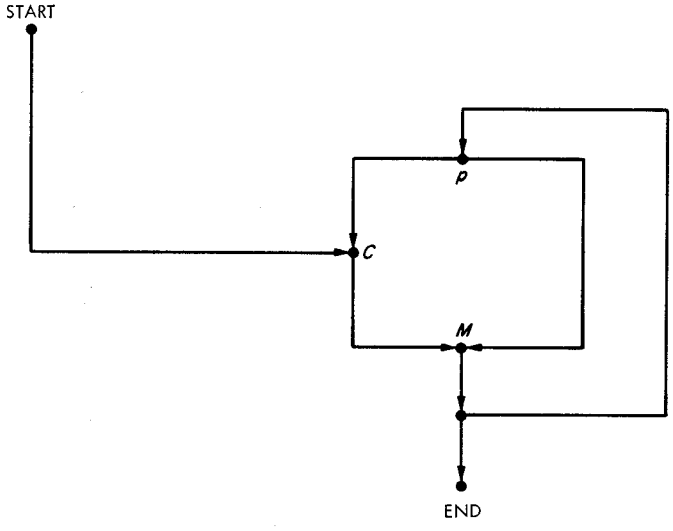

Fig. 8. The second case

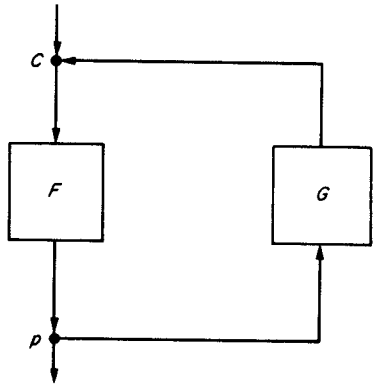

Fig. 9. LOOPREPEAT

LOOP COLLECTOR NODE
PERFORM *F*
EXIT LOOP IF *p*
PERFORM *G* (code for *G* ends with RETURN to COLLECTOR node.)

Fig. 10. Code for LOOPREPEAT

LOOP COLLECTOR NODE	LOOP COLLECTOR NODE
PERFORM *F*	PERFORM *G* UNLESS *p*
RETURN TO COLLECTOR NODE UNLESS *p*	(Code for *G* ends with RETURN to COLLECTOR node)
EXIT LOOP	EXIT LOOP
DOWHILE	DOUNTIL

Fig. 11. Code for DOWHILE and DOUNTIL